

# On Store Languages of Language Acceptors<sup>☆</sup>

Oscar H. Ibarra<sup>a,1</sup>, Ian McQuillan<sup>b,2</sup>

<sup>a</sup>Department of Computer Science

University of California, Santa Barbara, CA 93106, USA

<sup>b</sup>Department of Computer Science, University of Saskatchewan  
Saskatoon, SK S7N 5A9, Canada

---

## Abstract

It is well known that the “store language” of every pushdown automaton — the set of store configurations (state and stack contents) that can appear as an intermediate step in accepting computations — is a regular language. Here many models of language acceptors with various data structures are examined, along with a study of their store languages. For each model, an attempt is made to find the simplest model that accepts their store languages. Some connections between store languages of one-way and two-way machines generally are demonstrated, as with connections between nondeterministic and deterministic machines. A nice application of these store language results is also presented, showing a general technique for proving families accepted by many deterministic models are closed under right quotient with regular languages, resolving some open questions (and significantly simplifying proofs for others that are known) in the literature. Lower bounds on the space complexity for recognizing store languages for the languages to be non-regular are obtained.

*Keywords:* Store Languages, Turing Machines, Data Structures, Right Quotient, Automata

---

## 1. Introduction

A store configuration of a one-way or two-way language acceptor consists of the state followed by the contents of its memory (store) structure. It does not include the input and the position of the input head. For example, for a nondeterministic pushdown automaton (NPDA), a store configuration is represented by a string  $qx$ , where  $q$  is a state and  $x$  is the contents of the pushdown stack. For multi-tape acceptors, such as for an NPDA augmented with  $k$  reversal-bounded counters (NPCM) [1], the store configuration is represented by the string  $qxc_1^{j_1} \cdots c_k^{j_k}$ , where  $j_i$  represents the value of counter  $i$  in unary notation, and the  $c_i$  symbols and the symbols of  $x$  are disjoint. For a machine  $M$ , let  $S(M)$  be the set of store configurations that can appear as an intermediate step in accepting computations of  $M$ .

It is well-known that  $S(M)$  is a regular language for any NPDA  $M$  [2]. This result was key to showing properties of NPDAs, e.g., in deciding whether the set of all infixes (subwords) of a reversal-bounded NPDA is equal to  $\Sigma^*$  (i.e., is dense) [3]. In this paper, the store languages of several models of language acceptors are studied. Results that generalize (in often non-obvious ways) the aforementioned result concerning NPDAs are obtained, such as the following:

1. The following automata models with one-way read-only input, all produce regular store languages:  
 $k$ -flip pushdown automata [4] (which are like pushdown automata but can flip the pushdown store

---

<sup>☆</sup>©2016. This manuscript version is made available under the CC-BY-NC-ND 4.0 license  
<http://creativecommons.org/licenses/by-nc-nd/4.0/>

URL: [ibarra@cs.ucsb.edu](mailto:ibarra@cs.ucsb.edu) (Oscar H. Ibarra), [mcquillan@cs.usask.ca](mailto:mcquillan@cs.usask.ca) (Ian McQuillan)

<sup>1</sup>Supported, in part, by NSF Grant CCF-1117708 (Oscar H. Ibarra).

<sup>2</sup>Supported, in part, by a grant from Natural Sciences and Engineering Research Council of Canada (Ian McQuillan).

up to  $k$  times), reversal-bounded queue automata, nondeterministic Turing machines with a reversal-bounded worktape, and stack automata [5, 6]. The result for stack automata was shown recently [7] and so our result becomes an alternate proof that follows from existing results in the literature, and a new general and simple method demonstrated here for translating results from two-way to one-way machines.

2. The store language of a finite-crossing<sup>3</sup> two-way nondeterministic machine with reversal-bounded counters can be accepted by a one-way deterministic machine with reversal-bounded counters (DCM).
3. There is a non-finite-crossing two-way deterministic finite automaton with only one reversal-bounded counter whose store language cannot be accepted by any NPCM.
4. Some machine models (e.g., deterministic pushdown automata with reversal-bounded counters, DPCM) cannot accept their own store languages. This shows, as a corollary, that (2) above cannot be strengthened to make NCM a DCM.

NPCMs and NCMs have been extensively studied since their introductions in [1, 8]. They have found applications in areas such as timed-automata [9], model-checking and verification [10, 11], membrane computing [12], and Diophantine equations [13].

General connections between arbitrary types of one-way and two-way machines are also made, which is the key step to providing an alternate proof of regularity of the stack automata store languages. Connections between one-way nondeterministic and deterministic machines generally are also found.

An interesting application is presented showing the closure generally of many families of languages accepted by deterministic machines under right quotient with regular languages. Some of these resolve open problems in the literature, and others simplify existing known proofs. These include deterministic stack automata (known with a lengthy proof in [14]), deterministic  $k$ -flip pushdown automata (stated as an unresolved open problem in [15]), certain types of deterministic Turing machines, deterministic checking stack automata, deterministic reversal-bounded queue automata, and also providing an alternate proof for deterministic pushdown automata [16]. This general closure is somewhat surprising given the determinism and the nondeterministic nature of deletion occurring with quotients.

Finally, lower bounds on the space complexity for recognizing the store languages of space-bounded Turing machines for the languages to be non-regular are obtained.

## 2. Notation

An alphabet  $\Sigma$  is a set of symbols (usually assumed to be finite unless stated otherwise). The set of all words over  $\Sigma$  is denoted by  $\Sigma^*$ . Then a *language*  $L$  over  $\Sigma$  is any subset of  $\Sigma^*$ . Given a word  $w \in \Sigma^*$ , the *length* of  $w$  is denoted by  $|w|$ . Given  $a \in \Sigma$ , then  $|w|_a$  is the number of  $a$ 's in  $w$ . The *empty word* is denoted by  $\epsilon$ . The *reverse* of a word  $w$  is denoted by  $w^R$ , extended to the reverse  $L^R$  of a language  $L$ . Given two languages  $L_1, L_2$ , the *left quotient* of  $L_2$  by  $L_1$ ,  $L_1^{-1}L_2 = \{y \mid xy \in L_2, x \in L_1\}$ , and the *right quotient* of  $L_1$  by  $L_2$  is  $L_1L_2^{-1} = \{x \mid xy \in L_1, y \in L_2\}$ . The commutative closure of  $L$  is  $\{w \mid \exists v \in L, |v|_a = |w|_a, \forall a \in \Sigma\}$ . A language  $L \subseteq \Sigma^*$  is *letter-bounded* if there exists (not necessarily distinct)  $a_1, \dots, a_l \in \Sigma$  such that  $L \subseteq a_1^* \dots a_l^*$ . A language  $L$  is *bounded* if there exists  $w_1, \dots, w_l \in \Sigma^*$  such that  $L \subseteq w_1^* \dots w_l^*$ . Given two words  $u, v \in \Sigma^*$ ,  $u$  is a *prefix* of  $v$  if  $v = ux$ , for some  $x \in \Sigma^*$ ,  $u$  is a *suffix* of  $v$  if  $v = xu$  for some  $x \in \Sigma^*$ ,  $u$  is an *infix* of  $v$  if  $v = xuy$ , for some  $x, y \in \Sigma^*$ , and  $u$  is a *subsequence* of  $v$  if  $v = x_0u_1x_1 \dots x_{n-1}u_nx_n, x_0, \dots, x_n, u_1, \dots, u_n \in \Sigma^*, u = u_1 \dots u_n$ .

In this paper, introductory knowledge of automata and formal languages is assumed (see [17] for an introduction), including finite automata (NFAs and DFAs), pushdown automata (NPDAs), Turing machines (NTMs and DTMs), and generalized sequential machines (gsms).

---

<sup>3</sup>Finite-crossing means that the input head crosses the boundary of any two adjacent input symbols at most a fixed number of times.

### 3. Store Languages of One-Way Machines

Many different kinds of machine models are studied in this paper, such as finite automata, pushdown automata [17], reversal-bounded multicounter machines [1], stack automata (similar to a pushdown automata with the ability to read, but not change on the inside of the pushdown) [5, 6], Turing machines [17], queue automata [18], flip-pushdown automata (machines with the ability to flip the pushdown at most  $k$  times) [4] and also combinations of their stores within individual machines. The store language of each depends on the precise definition of each type of machine. It is possible to define all such models generally by varying the “store type” similar to Abstract Families of Automata [19] or storage types [20], and then the store language only needs to be defined once for all types of machines. This approach is followed here due to the large number of machine models considered, because it allows to make general connections between types of machines, and because store languages depend considerably on the precise definition of the machines.

**Definition 1.** A store type is a tuple  $\Omega = (\Gamma, I, f, g, c_0, L_I)$ , where

- $\Gamma$  is the set of store symbols (potentially infinite, available to all machines using this store),
- $I$  is the set of instructions,
- $g$  is the read function, a partial function from  $\Gamma^*$  to  $\Gamma$ ,
- $f$  is the write function, a partial function from  $\Gamma^* \times I$  to  $\Gamma^*$ ,
- $c_0 \in \Gamma^*$  is the initial store configuration,
- $L_I \subseteq I^*$  is the instruction language.

Thus, a store type defines a type of auxiliary store. The write function  $f$  indicates how each store contents change in response to each instruction, and the read function indicates how machines read from each store contents. Every machine using this store type starts with  $c_0$  on its store. Lastly,  $L_I$  is a type of filter that can restrict the allowable sequences of instructions. This is useful for several purposes, such as defining reversal-bounded store types.

One example of a classical store type is given next, followed by the definition of the set of all machines using store types.

**Example 1.** The pushdown store type is  $\Omega = (\Gamma, I, f, g, c_0, L_I)$  where  $c_0 = Z_0 \in \Gamma$ ,  $\Gamma_0 = \Gamma - \{Z_0\}$  ( $Z_0$  is the bottom-of-stack marker),  $I = \Gamma^*$ ,  $L_I = I^*$  (ie. there is no restriction as to the possible sequences of instructions),  $g(xa) = a, x \in \Gamma^*, a \in \Gamma$ ,  $f(xa, y) = xy$ , where  $xa, xy \in Z_0\Gamma_0^*$ .

The machines (defined next) using this store type are equivalent to standard pushdown automata [17].

**Definition 2.** Given store types  $\Omega_1, \dots, \Omega_k$  with  $\Omega_i = (\Gamma_i, I_i, f_i, g_i, c_{0,i}, L_{I,i}), 1 \leq i \leq k$ , where  $\Gamma_i$  are pairwise disjoint, for  $1 \leq i \leq k$ , a one-way nondeterministic  $k$ -tape  $(\Omega_1, \dots, \Omega_k)$ -machine is a tuple  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ , where  $Q$  is the finite set of states,  $\Sigma$  is the input alphabet,  $q_0 \in Q$  is the initial state,  $F \subseteq Q$  is the set of final states,  $\Gamma$  is a finite subset of  $\Gamma_1 \cup \dots \cup \Gamma_k$  and the finite transition relation  $\delta$  is from  $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma_1 \times \dots \times \Gamma_k$  to  $Q \times I_1 \times \dots \times I_k$ .

Then a configuration of  $M$  is a tuple  $(q, w, \gamma_1, \dots, \gamma_k)$ , where  $q \in Q$  is the current state,  $w \in \Sigma^*$  is the remaining input, and  $\gamma_i \in \Gamma_i^*$  is the contents of the  $i$ th store, for  $1 \leq i \leq k$ . The derivation relation  $\vdash_M$  is defined by:  $(q, aw, \gamma_1, \dots, \gamma_k) \vdash_M (q', w, \gamma'_1, \dots, \gamma'_k), w \in \Sigma^*, a \in \Sigma \cup \{\epsilon\}, \gamma_i, \gamma'_i \in \Gamma_i^*, 1 \leq i \leq k, q, q' \in Q$ , if there exists

$$(q', \iota_1, \dots, \iota_k) \in \delta(q, a, d_1, \dots, d_k), \quad (1)$$

such that  $g_i(\gamma_i) = d_i, f(\gamma_i, \iota_i) = \gamma'_i$ , for all  $i, 1 \leq i \leq k$ . This is extended to  $\vdash_M^*$ , the reflexive and transitive closure of  $\vdash_M$ . Sometimes, bijective labels  $T$  will be associated with transitions of  $M$ , and in such cases, the derivation relation using transition  $t$  is sometimes written as  $\vdash_M^t, t \in T$ , generalized to words  $\vdash_M^x, x \in T^*$ .

For  $1 \leq i \leq k$ , define a homomorphism  $\pi_i$  from  $T^*$  to  $I_i^*$  where  $\pi_i(t) = \iota_i$  for  $t$  of the form of (1). Then  $x$  is valid if  $\pi_i(x) \in L_{I,i}$ , for each  $i$ ,  $1 \leq i \leq k$ .

The language accepted by  $M$ ,  $L(M) = \{w \mid (q_0, w, c_{0,1}, \dots, c_{0,k}) \vdash_M^x (q_f, \epsilon, \gamma_1, \dots, \gamma_k), q_f \in F, w \in \Sigma^*, \gamma_i \in \Gamma_i, 1 \leq i \leq k, x \in T^* \text{ is valid}\}$ .

The store language of  $M$ ,

$$S(M) = \{q\gamma_1 \dots \gamma_k \mid (q_0, uv, c_{0,1}, \dots, c_{0,k}) \vdash_M^x (q, v, \gamma_1, \dots, \gamma_k) \vdash_M^y (q_f, \epsilon, \gamma'_1, \dots, \gamma'_k), \\ q_f \in F, u, v \in \Sigma^*, \gamma_i, \gamma'_i \in \Gamma_i, 1 \leq i \leq k, xy \in T^* \text{ is valid}\}.$$

Thus,  $S(M)$  is the set of store configuration representatives that can appear as an interim step of an accepting computation. It is also enforced that, if  $k > 1$ , then  $\Gamma_1, \dots, \Gamma_k$  are all disjoint. By a slight abuse of notation, even if machines with more than one tape that are the same store type are examined, it is assumed that the stores use disjoint alphabets. Note that since the letters used in each component are disjoint, when reading a string of  $S(M)$ , it is possible to know which of the  $k$  stores is being read.

For an  $\Omega$  store type, a  $k$ - $\Omega$  machine is a  $k$ -tape machine where each tape is of type  $\Omega$ .

**Definition 3.** Given a set of machines  $\mathcal{M}$ , then let  $\mathcal{L}(\mathcal{M})$  be the family of languages accepted by  $\mathcal{M}$ . Also, let  $\mathcal{S}(\mathcal{M})$  be the family of store languages of machines in  $\mathcal{M}$ .

Some types of one-way deterministic automata will also be examined in this paper. In those circumstances, all machines are defined to scan input  $w\triangleleft$ , where  $w \in \Sigma^*$ , and  $\triangleleft$  is the right end-marker (this is needed for some types of machines such as DCM [21]). Then, a machine is deterministic if  $|\delta(q, a, d_1, \dots, d_k) \cup \delta(q, \epsilon, d_1, \dots, d_k)| \leq 1$  for all  $q \in Q, a \in \Sigma \cup \{\triangleleft\}, d_i \in \Gamma_i$ , the language accepted by  $M$ ,  $L(M) = \{w \mid (q_0, w\triangleleft, c_{0,1}, \dots, c_{0,k}) \vdash_M^x (q_f, \epsilon, \gamma_1, \dots, \gamma_k), q_f \in F, w \in \Sigma^*, \gamma_i \in \Gamma_i^*, 1 \leq i \leq k, x \text{ is valid}\}$ , and  $S(M) = \{q\gamma_1 \dots \gamma_k \mid (q_0, w\triangleleft, c_{0,1}, \dots, c_{0,k}) \vdash_M^x (q, w', \gamma_1, \dots, \gamma_k) \vdash_M^y (q_f, \epsilon, \gamma'_1, \dots, \gamma'_k), q_f \in F, w, w' \in \Sigma^*, \gamma_i, \gamma'_i \in \Gamma_i^*, 1 \leq i \leq k, xy \text{ is valid}\}$ .

Given store types  $\Omega_1, \dots, \Omega_k$ , the set of all one-way nondeterministic, or deterministic,  $(\Omega_1, \dots, \Omega_k)$ -machines that can be built using this store type will be examined.

In this notation, store types that are equivalent to standard automata models from the literature will be described. Indeed, pushdown automata are machines that start with  $Z_0$  on the pushdown, can read the top of the pushdown on its transitions, and replace the topmost letter with a word. These correspond with  $\Omega$ -machines as built with the pushdown store type of Example 1. Let NPDA be the set of all such pushdown automata.

An  $l$ -reversal-bounded pushdown store is the same except  $L_I$  is set to the concatenation of  $l$  alternating sequences of  $(\{y \mid y \in \Gamma^*, |y| \geq 1\})^*$  and  $(\{y \mid y \in \Gamma^*, |y| \leq 1\})^*$  ie. there are at most  $l$  alternations between increasing and decreasing the size of the stack.

A counter store type restricts a pushdown store to having single symbol  $c \in \Gamma_0$  (plus  $Z_0$ ). At each step, essentially based on whether the counter is empty or non-empty, a machine can change each counter by  $+1, 0$ , or  $-1$ . Similarly, an  $l$ -reversal-bounded counter can be defined.

One can also define store types for  $k$   $l$ -reversal-bounded counters. The set of all machines that have  $k$   $l$ -reversal-bounded counters, for some  $k, l \geq 1$  is denoted by NCM. Note that NCM is a union of sets of machines that can be built using store types.

**Example 2.** A queue store type is a tuple  $\Omega = (\Gamma, I, f, g, c_0, L_I)$ , where  $I = \{\text{enqueue}(y) \mid y \in \Gamma^*\} \cup \{\text{dequeue}\}$ ,  $c_0 = \epsilon$ ,  $L_I = I^*$ ,  $g(x)$  is  $\epsilon$  if  $x = \epsilon$ , and the leftmost symbol of  $x$  otherwise,  $f(x, \text{dequeue}) = \Gamma^{-1}x$ , and  $f(x, \text{enqueue}(y)) = xy$ .

Stacks can be similarly defined to pushdowns, where there are also instructions to enter the inside of the stack in a read-only fashion (these require that the read head be included in the store language in a similar fashion to Turing tapes below).

**Example 3.** The  $k$ -flip-pushdown store type is a tuple  $\Omega = (\Gamma, I, f, g, c_0, L_I)$ , where  $c_0 = Z_0$ ,  $\Gamma_0 = \Gamma - \{Z_0\}$ ,  $I = \Gamma^* \cup \{\text{flip}\}$ ,  $L_I = I^*$ ,  $g(xa) = a, x \in \Gamma^*, a \in \Gamma$ ,  $f(xa, y) = xy$ , where  $xaxy \in Z_0\Gamma_0^*$ , and  $f(x, \text{flip}) = Z_0(Z_0^{-1}x)^R, x \in \Gamma^*$ , and  $L_I$  restricts at most  $k$  flip moves to be applied.

Machines defined with this type are equivalent to those in [4], although they are classically defined where the flips are performed with a separate function.

**Example 4.** A Turing store is a tuple  $\Omega = (\Gamma, I, f, g, c_0, L_I)$  where  $c_0 = \downarrow \sqcup$  ( $\downarrow$  is the r/w head, and  $\sqcup$  is the blank symbol, so each tape initially only has the r/w head followed by a blank,  $\Gamma_0 = \Gamma - \{\downarrow\}$ ,  $\Gamma_1 = \Gamma_0 - \{\sqcup\}$ ),  $I = \{a^{\leftarrow}, a^{\rightarrow}, a \mid a \in \Gamma_0\}$  (this groups together both the new symbol written in the current tape cell, and the direction for the head to move),  $L_I = I^*$ ,  $g(x \downarrow bx') = b, b \in \Gamma_0, x, x' \in \Gamma_0^*$ . Then, at each step, a machine with this store can read the symbol under the read/write head on each tape, and execute an instruction which corresponds to a standard TM instruction, write an  $a$  in the current cell and move left, right or stay. The write function is defined by, for all  $x \in \Gamma_1 \Gamma_0^* \cup \{\epsilon\}, x' \in \Gamma_0^* \Gamma_1 \cup \{\epsilon\}$ :

$$\begin{aligned}
& \bullet f(x \downarrow bx'), a) = x \downarrow ax', \\
& \bullet f(x \downarrow bx'), a^{\leftarrow}) = \begin{cases} \downarrow \sqcup ax' & \text{if } ax' \notin \sqcup^* \text{ and } x = \epsilon, \\ x_1 \downarrow cax' & \text{if } ax' \notin \sqcup^* \text{ and } x = x_1 c, c \in \Gamma_0, \\ \downarrow \sqcup & \text{if } ax' \in \sqcup^* \text{ and } x = \epsilon, \\ x_1 \downarrow c & \text{if } ax' \in \sqcup^* \text{ and } x = x_1 c, c \in \Gamma_0, \end{cases} \\
& \bullet f(x \downarrow bx'), a^{\rightarrow}) = \begin{cases} xa \downarrow \sqcup & \text{if } xa \notin \sqcup^* \text{ and } x' = \epsilon, \\ xa \downarrow x' & \text{if } xa \notin \sqcup^* \text{ and } x' \neq \epsilon, \\ \downarrow \sqcup & \text{if } xa \in \sqcup^* \text{ and } x' = \epsilon, \\ \downarrow x' & \text{if } xa \in \sqcup^* \text{ and } x' \neq \epsilon. \end{cases}
\end{aligned}$$

Then, a machine with one such store type is a Turing machine with a one-way read-only input tape, and one read/write store tape. The store starts off empty (the read/write head followed by a blank), and they can extend in both directions as symbols are added to the left and right. They can also shrink in size if everything to the right of the read/write head is a blank, as with the left. This is exactly how configurations of Turing machines change [17].

Furthermore,  $l$ -reversal-bounded 1-tape Turing stores can be defined by restricting  $L_I$  so that the number of alternations between moving right and left on the tape is at most  $l$ .

Let  $\mathcal{L}(\text{REG})$  be the family of languages accepted by NFAs. Also define machines with one pushdown for the first tape, and  $k$  additional reversal-bounded counters, where each word in the store language is of the form  $qxc_1^{j_1} \cdots c_k^{j_k}$ , where  $q$  is a state,  $x$  is the contents of the pushdown, and  $j_1, \dots, j_k$  are the contents of the counters. Let NPCM be the set of machines with one pushdown, and some number  $k$  of counters where the pushdown is unrestricted, but the counters are reversal-bounded. The family of languages accepted by NPCM [1, 22] is of interest since it has a decidable emptiness and membership problem, and only accepts semilinear languages.

Let NQA be the set of queue automata [18]. As with NPCM, define machines with one queue for the first tape, and  $k$  additional counters. Let NQCM be the set of machines with one queue, and some number  $k$  of counters where the counters are reversal-bounded (if the queue is also reversal-bounded, these only accept semilinear languages [18], otherwise they have the same power as Turing machines). Let NSA be the set of stack automata [6, 5]. Also, define machines with one stack for the first tape, and  $k$  additional counters. Let NSCM be the set of machines with one stack, and some number  $k$  of counters where the counters are reversal-bounded (if the stack is reversal-bounded, this implies that there is also a bound on the number of changes in direction of the read head when it reads inside the stack structure). Let NFPA be the set of  $k$ -flip pushdown machines, for some  $k$  [4]. Replacing N with D gives each deterministic variant.

### 3.1. Store Languages of Turing Machines and Other One-Way Automata Models

Store languages have already been investigated for nondeterministic pushdown automata. It has been shown [2] that the store language of each NPDA is a regular language. Moreover, the proof contains an effective construction.

**Proposition 4.** [2] *Given a one-way NPDA  $M$ ,  $S(M)$  is a regular language, and  $\mathcal{S}(\text{NPDA}) \subseteq \mathcal{L}(\text{REG})$ .*

First, a general decidability proposition is proved for machine models where the emptiness problem is undecidable.

**Proposition 5.** *Given a set of machines  $\mathcal{M}$  defined using (potentially multiple) store types, such that the emptiness problem is undecidable for  $M \in \mathcal{M}$ . Then the problem, given  $M \in \mathcal{M}$  and a word  $x$ , determine whether  $x \in S(M)$ , is undecidable.*

PROOF. Let  $M \in \mathcal{M}$  be a machine with initial state  $q_0$  and initial store contents  $z$  (which can be the concatenation of multiple initial store contents for multi-store machines). Then  $q_0z$  is in the store language of  $M$  if and only if  $L(M)$  is not empty.

Hence, membership in  $S(M) \in \mathcal{M}$  is undecidable.  $\square$

This is true for sets of one-way machines, and also two-way machines investigated later in the paper. And in fact, it even holds for complexity classes, such as deterministic Turing machines with a one-way read-only input tape and a logspace bounded worktape (the store). These have a decidable membership problem but an undecidable emptiness problem. Despite the languages accepted by these machines being recursive (and in P), membership in the store language is undecidable (and so there cannot be an effective construction to accept the store languages with another model, such as any model with a decidable membership problem).

Next, store languages of restricted NTMs will be studied. They will be especially useful for characterizing store languages of other machine models. In particular, NTMs with a one-way read-only input tape and one reversal-bounded read/write worktape are considered. In terms of languages accepted, these machines are powerful enough to simulate a number of different machine models exactly, such as one-way nondeterministic reversal-bounded pushdown automata, reversal-bounded queue automata, reversal-bounded stack automata, and reversal-bounded  $k$ -flip pushdown automata, where the worktape acts exactly like the other stores. Then, since the direction that the store changes is the same between each reversal, the Turing machine read/write head only needs to make a bounded number of changes in direction.

Next, the store languages of these Turing machines are examined.

**Proposition 6.** *Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  be an NTM with a one-way read-only input tape and a reversal-bounded read/write worktape. Then  $S(M) \in \mathcal{L}(\text{REG})$ .*

PROOF. Let  $M$  make at most  $l$  reversals on the worktape. Note that  $L(M) \subseteq \Sigma^*$ , and  $S(M) \subseteq Q\Gamma^*$ . Let  $\Gamma_0 = \Gamma - \{\downarrow\}$ , and  $\Gamma'_0 = \{a' \mid a \in \Gamma_0\}$ , a new alphabet (each letter is a “primed” version of a letter in  $\Gamma_0$ , including a primed version of the blank symbol  $\sqcup$ ). Define a new alphabet  $C$  whose symbols have “tracks”, with less than or equal to  $l + 2$ -tracks of the form  $(a_1, a_2, \dots, a_p), p \leq l + 2$  where  $a_1$  is in  $\Gamma_0 \cup \Gamma'_0 \cup Q$  and  $a_i \in \Gamma_0 \cup \Gamma'_0$ , for each  $i, 2 \leq i \leq p$ .

An intermediate 2-way NFA  $M'$  is constructed, whose input is in  $C^*$  delimited with end-markers  $\triangleright$  and  $\triangleleft$ . Thus the input to  $M'$  is  $\triangleright w \triangleleft$ , where  $w \in C^*$ . The input  $w$  can be thought of as having less than or equal to  $l + 2$  tracks.

Intuitively,  $M'$  is trying to verify that the contents of the first track represents a configuration in an accepting computation of  $M$ , where a symbol  $a' \in \Gamma'_0$  is used in place of  $\downarrow a$  in the store language. To do this,  $M'$  nondeterministically guesses an input  $x \in \Sigma^*$  and simulates  $M$  on  $x$ , track 2 is verified to be the initial store contents, each track from tracks 3 to  $p - 1$  is verified to be the store contents at a point of reversal, and track  $p$  is verified to be a final accepting configuration. All tracks are padded by blank symbols to all be of the same length.

Then  $M'$  operates as follows:

1. The first track is verified to contain a word  $\sqcup^n q w' \sqcup^l$ , where  $q \in Q, n, l \geq 0, w' \in \Gamma_0^* \Gamma'_0 \Gamma_0^*$  that does not start or end with  $\sqcup$  (below,  $M'$  will verify that the word obtained from  $qw'$  by replacing  $a'$  with  $\downarrow a$  is in  $S(M)$ ).



2.  $M'$  goes to the left end-marker  $\triangleright$ .  $M'$  checks that the second track contains  $\sqcup^m \sqcup' \sqcup^r$  for some  $m, r \geq 0$  (the worktape starts off with only blanks, it is implied that all tracks are of length  $m + r + 1 = n + l + |w'| + 1$ ).
3.  $M'$  then simulates the TM  $M$  on a guessed input  $x \in \Sigma^*$ , letter-by-letter, but instead of writing, verifies the next track contents is an updated version of the current track at the next point of reversal. Between the initial configuration and the first reversal, between every two reversals, and between the last reversal and the final configuration,  $M'$  checks that the contents of track  $i + 1$  is the updated contents of the worktape from the worktape on track  $i$  for  $i = 2, \dots, p - 1$ . To do this, if track  $i$  contains  $\sqcup^\alpha x a' x' \sqcup^\beta$ ,  $x, x' \in \Gamma_0^*$ ,  $a \in \Gamma_0'$ , and say  $i$  is even (the case is similar if  $i$  is odd), then it is verified that track  $i + 1$  starts with  $\sqcup^\alpha x$ . Then, if  $M$  uses a transition that replaces  $a$  with  $b$  and moves right on the worktape, then track  $i + 1$  has  $b$  next (a sequence of transitions that stays on the same storage cell are remembered in the finite control), and this simulation continues until  $M$  makes a reversal. When  $M$  makes a reversal, say from moving right to left,  $M'$  first “marks” the point of reversal by reading a primed symbol in that position of track  $i + 1$  (storing the read/write head in track  $i + 1$ ), then it moves to the right end-marker to check that each symbol in track  $i$  from the point of reversal to the right end-marker match the symbols in track  $i + 1$  (this also implies that track  $i + 1$  has exactly one symbol from  $\Gamma_0'$ ).  $M'$  then moves left back to the point of reversal (which is retrievable from the primed symbol), and resumes the simulation using the next track from the current track.
4. At some nondeterministically guessed reversal of the simulation as described in step 3 (say while scanning track  $i$ , and track  $i + 1$  reverses from left to right), while  $M'$  is verifying that track  $i + 1$  follows from track  $i$ , in parallel,  $M'$  verifies that the contents of track 1 is a configuration of the Turing machine between these two reversals. To do this,  $M'$  remembers the state  $q$  on track 1, then compares track 1 to track  $i + 1$  symbol-by-symbol, until reaching the read/write head in track 1, where the current state of the simulated machine is verified to be  $q$  and the remaining part of track 1 is verified to be the same as track  $i$ . Thus, track 1 is a configuration between tracks  $i$  and  $i + 1$ .  $M'$  then continues the simulation as in step 3.

Since  $M'$  can only read and not write (on the input), it just verifies the moves and that the changes in the symbols of the TM  $M$  on track  $i$  are reflected in the  $i + 1$ st track.  $M'$  accepts if  $M$  accepts.

It is known that 2-way NFAs accept only regular languages [17]. Then, apply a gsm [17] to extract just the word  $w'$  from the first track, erasing blanks appropriately, and replacing any symbol  $a' \in \Gamma'$  by  $\downarrow a$ . Since regular languages are closed under gsm mappings, the result follows.  $\square$

Next, consider NTMs augmented with reversal-bounded counters. The proof uses 2NCM, which are two-way nondeterministic reversal-bounded multicounter machines, and a similar technique as the proof of Proposition 6, ultimately determining that the store languages are accepted by one-way NCM machines.

**Proposition 7.** *Let  $M$  be an NTM with a one-way read-only input tape, a reversal-bounded read/write worktape, and  $k$  reversal-bounded counters (hence the store consists of the read/write tape followed by the values of the counters). Then  $S(M) \in \mathcal{L}(\text{NCM})$ .*

PROOF. The proof of Proposition 6 is generalized (using the same alphabets).

Construct an intermediate 2NCM  $M'$  that is reversal-bounded on the input tape, with  $2k$  reversal-bounded counters.

1.  $M'$  will have as input  $z = wc_1^{i_1} \dots c_k^{i_k}$  (with end-markers) where  $w$  has multiple read-only tracks (over the alphabet  $C$ , just like in Proposition 6), and the first track is  $\sqcup^n q w' \sqcup^l$ ,  $q \in Q$ ,  $n, l \geq 0$ ,  $w' \in \Gamma_0^* \Gamma_0' \Gamma_0^*$  that does not start or end with  $\sqcup$ .
2.  $M'$  simulates  $M$ 's reversal-bounded read/write worktape on the tracks of the read-only  $w$  (as in Proposition 6) and using reversal-bounded counters to simulate the reversal-bounded counters of  $M$ . However,  $M'$  keeps two copies of each counter, where the two sets of counters are updated synchronously (and are therefore identical during the first part of the simulation).

3. At some point,  $M'$  nondeterministically guesses that the contents of track 1,  $qw'$  together with counter values  $(i_1, \dots, i_k)$  encoded in the input is a representation of a configuration between the current track and the next track. Then, as in Proposition 6,  $M'$  matches the symbols in the first track with track  $i + 1$  until the symbol from  $\Gamma'_0$  in track 1, and if so, stops updating one set of the counters. Then  $M'$  continues by matching track 1 with track  $i$ . Then the simulation of  $M$  continues, using the other set of counters and on the remaining tracks (as in Proposition 6). At the end of the simulation,  $M'$  verifies that the non-updated set of counters equals the input  $c_1^{i_1} \dots c_k^{i_k}$  (if this is the case, then  $qw'c_1^{i_1} \dots c_k^{i_k}$  is indeed an intermediate store configuration of an accepting computation), and if so, accepts if  $M$  accepts.

Now  $M'$  is a reversal-bounded (on the input) 2NCM. Hence,  $M'$  can be converted to an equivalent one-way NCM  $M''$ , i.e.,  $L(M'') = L(M')$  (this can be done even for the more general finite-crossing 2NCM) [23].

Since  $\mathcal{L}(\text{NCM})$  is a full trio [1], it is closed under gsm mappings. Hence, construct an NCM  $M'''$  that applies a gsm that extracts  $qw'c_1^{i_1} \dots c_k^{i_k}$  (and replaces  $a' \in \Gamma'$  with  $\downarrow a$ ) from the first track of  $w$  and  $z$ .

It follows that  $S(M)$  is in  $\mathcal{L}(\text{NCM})$ .  $\square$

From the results of Section 3.1 on store languages of Turing machines, it is possible to accept the store languages of other machine models (this does not follow directly from the fact that such Turing machines can simulate the input languages of other models, as store languages rather than input languages are of interest here. The first result is for reversal-bounded NQCM (with a queue and counters, all reversal-bounded).

**Proposition 8.** *If  $M$  be a reversal-bounded NQCM, then  $S(M) \in \mathcal{L}(\text{NCM})$ .*

PROOF. Given  $M$  with  $k$  counters, construct an intermediate NTM  $Z$  with an input tape plus one reversal-bounded read/write tape, and  $k$  additional reversal-bounded counters, whose resulting store language will be in  $\mathcal{L}(\text{NCM})$  by Proposition 7.  $Z$  operates as follows: every time  $M$  enqueues  $y = b_1 \dots b_m$ ,  $Z$  writes to the right end of the read/write worktape writing  $y$  and simulating the queue exactly. Every time  $M$  dequeues,  $Z$  removes characters from the left end of the tape towards the right (thus removing characters from the store of the Turing machine as well). Since the queue is reversal-bounded, there is a bounded number of switches between enqueueing and dequeuing.

Then, the store language of  $Z$ , has each word of the form  $qwc_1^{i_1} \dots c_k^{i_k}$  and is in  $\mathcal{L}(\text{NCM})$  by Proposition 7. Then  $S(Z)$  will be transformed into  $S(M)$  by a gsm  $g$ . The only differences between the store of such a Turing machine and a NQCM is the read/write head does not occur in the store language of the queue nor the blank symbol after the read/write head, whereas it does in the Turing machine, and the simulated state  $q'$  of  $M$  is not necessarily the same as the state  $q$  of  $Z$ . But the simulated state  $q'$  is determined by  $q$  in the simulation. Furthermore, if  $M$  enqueues more than one symbol ( $m > 1$ ), then  $Z$  requires  $m$  moves. Then all intermediate states used by  $Z$  when writing each  $b_i, i < m$  is not mapped by  $g$ . Hence, the store language of  $M$  is in  $\mathcal{L}(\text{NCM})$ .  $\square$

Note that in the proof of the above result, if there are no counters in  $M$ , then only a Turing machine with one reversal-bounded read/write worktape is required, whose store language is a regular language by Proposition 6.

**Corollary 9.** *If  $M$  is a one-way reversal-bounded queue automaton, then  $S(M) \in \mathcal{L}(\text{REG})$ .*

The same result on NTMs could be used to show that all reversal-bounded NPCM store languages are in  $\mathcal{L}(\text{NCM})$  but this result will be improved later in the paper.

However, next it will be shown that the same is true for reversal-bounded stack automata augmented by reversal-bounded counters. Recall that stack automata can operate like pushdown automata with additional instructions,  $-1, 0, +1$  that can move left, stay or move right in the pushdown store, in a read-only fashion [5].

**Proposition 10.** *Let  $M$  be a reversal-bounded NSCM or a reversal-bounded NPCM. Then  $S(M) \in \mathcal{L}(\text{NCM})$ .*



PROOF. Let  $M$  be a reversal-bounded stack automaton with  $k$  counters. The stack can be simulated faithfully by a read/write worktape of a Turing machine that is also reversal-bounded (and here, the read/write head is kept in the Turing machine as it occurs in the stack store as well). Similarly the counters are simulated exactly as well. Then  $S(M)$  is an  $\mathcal{L}(\text{NCM})$ . Similarly for NPCMs.  $\square$

This immediately implies that every reversal-bounded stack automaton has a regular store language, but this result too will be improved later in the paper.

The store languages of NPCM generally will be considered in a followup paper.

Next, a  $k$ -flip-pushdown automaton is a pushdown automaton, with the ability to flip its store. This can happen at most  $k$  times in an accepting computation. Despite the additional ability to flip the store, regularity of the store language is preserved.

**Proposition 11.** *Given a one-way  $k$ -flip pushdown automaton  $M$ ,  $k \geq 0$ , then  $S(M) \in \mathcal{L}(\text{REG})$ .*

PROOF. For a  $k$ -flip NPDA (or an NPDA)  $M$  with state  $q$ , let  $\text{Acc}_q(q) = \{x \mid (q_0, w, Z_0) \vdash_M^* (q, \epsilon, x)\}$  and  $\text{co-Acc}(q) = \{x \mid (q, w, x) \vdash_M^* (q_f, \epsilon, x'), q_f \in F\}$ . It is known that for NPDAs  $M$ , both  $\text{acc}(q)$  and  $\text{co-Acc}(q)$  are regular [2], and therefore  $S(M)$  is as well.

Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  be a  $k$ -flip pushdown automaton.

Let  $z = p_0, p_1, \dots, p_{2l}, p_{2l+1}$  be a sequence of states where  $l \leq k, p_0 = q_0, p_{2l+1} \in F$ , and there is a ‘flip’ transition that switches states from  $p_{2i-1}$  to  $p_{2i}$ , for all  $1 \leq i \leq l$ . Consider a flip pushdown  $M_z$  (associated with each such sequence  $z$ ) where  $M_z$  acts exactly like  $M$  but flips in order from  $p_1$  to  $p_2$ , and so on up to, flips from  $p_{2l-1}$  to  $p_{2l}$  at exactly these states,  $p_{2i-1}, 1 \leq i \leq l$ , transitioning to  $p_{2i}$ . Then, for  $q, q' \in Q$ , let  $M_{q,q'}$  be a pushdown automaton with  $q$  as initial state, and  $q'$  as final state, with no flip transitions, but otherwise acts like  $M$ .

Let  $p \in Q$ . It will be shown that  $\text{Acc}_{M_z}(p)$  is a regular language. Let  $i$  be such that  $0 \leq i \leq l$ . Then the goal is to accept all words that can appear in the store of  $M$  when  $M$  is in state  $p$  and between states  $p_{2i}$  and  $p_{2i+1}$ . Make a pushdown automaton  $M_{p_0, p_1}$ . Indeed,  $\text{Acc}(M_{p_0, p_1})$  is regular. Then the set of words that can appear on the pushdown in the computation at state  $p_1$  is a regular language ( $\text{Acc}_{M_{p_0, p_1}}(p_1)$ ). Then, the reversal of this language is regular as well,  $X_1$  (by first removing the bottom-of-stack marker  $Z_0$ , taking the reversal, and re-concatenating  $Z_0$ ). Then build an NPDA that starts by nondeterministically pushing a word of  $X_1$  and then simulates the NPDA  $M_{p_2, p_3}$ . Then, the set of words that can appear in the store language is again regular, and so the reversal  $X_2$  is as well. Continue until considering the pushdown  $M_{p_{2l}, p_{2l+1}}$ . Then take the set of all words that can appear in the pushdown starting and ending in state  $p$ . This must be regular as well. By taking the union over all  $i$  and all  $z$ ,  $\text{Acc}(p)$  in  $M$  must be regular as well.

In the opposite direction, consider the pushdown  $M_{p_{2l}, p_{2l+1}}$ . Then  $\text{co-Acc}_{M_{p_{2l}, p_{2l+1}}}(p)$  is a regular language, and so the reverse  $X_l$  is regular. Create a pushdown that simulates  $M_{p_{2(l-1)}, p_{2l+1}}$ , but only accepts if at the end, the store is in  $X_l$ . Continuing in this way, it is possible to compute  $\text{co-Acc}_M(p)$  as well, and it is regular. Hence,  $S(M)$  is regular.  $\square$

Next, the store languages of NCMs are analyzed. Surprisingly, only deterministic machines are needed to accepted them.

**Proposition 12.** *If  $M$  is an NCM, then  $S(M) \in \mathcal{L}(\text{DCM})$ . Thus,  $S(\text{NCM}) \subseteq \mathcal{L}(\text{DCM})$ .*

PROOF. Let  $M$  have counters  $C_1, \dots, C_k$ . First, construct NCM  $M'$  with counters  $C_1, \dots, C_k, D_1, \dots, D_k$ . Then  $M'$ , when given an input  $z$ , checks that  $z$  is of the form  $qc_1^{i_1} \dots c_k^{i_k}$  for  $q$  a state,  $i_1, \dots, i_k \geq 0$  (this can be done by a DFA which can run in parallel to the checking that  $z$  is in  $S(M)$ ). To check that  $z$  is in  $S(M)$ ,  $M'$  on transitions that do not read any input, guesses an input  $x$  to  $M$  (i.e., it guesses the symbols comprising  $x$ ) and simulates  $M$  using counters  $C_1, \dots, C_k$  and  $D_1, \dots, D_k$  (i.e.,  $D_1, \dots, D_k$  are duplicate counters which operate like  $C_1, \dots, C_k$ ). At some point (nondeterministically chosen),  $M'$  stops updating counters  $D_1, \dots, D_k$  and remembers the current state  $q'$  but continues the simulation with counters  $C_1, \dots, C_k$ . When  $M$  accepts,  $M'$  checks that the value of  $D_j$  is  $i_j$ , for all  $j$ ,  $1 \leq j \leq k$  and that  $q = q'$ . The NCM  $M'$  can then be converted to a DCM  $M''$ , since it is known that any NCM accepting a bounded language can be accepted by a DCM [24].  $\square$

**Proposition 13.** *Let  $M$  be an NCM which accepts with all counters zero and in a unique accepting state  $f$  which is never re-entered. Let  $L = \{fZ_0x \mid x \in L(M)\}$ . Then there is a 0-reversal-bounded DPCM  $M'$  and a regular language  $R$  such that  $L = S(M') \cap R$ . Then  $L(M) = (fZ_0)^{-1}S(M')$ .*

PROOF. Let  $M$  be an NCM with  $k$  counters and input alphabet  $\Sigma$ . Represent each transition of  $M$  by an abstract symbol:

$$[(q, a, s_1, \dots, s_k) \rightarrow (p, d_1, \dots, d_k)],$$

where  $p$  and  $q$  are states,  $a$  is either in  $\Sigma$  or  $\epsilon$ ,  $s_i$  represents the status (zero or non-zero) of counter  $i$  and  $d_i$  is the change in counter  $i$ . Let  $\Delta$  be the set of symbols representing the transitions.

The input alphabet of the DPCM  $M'$  is  $\Delta$ . For a string  $y \in \Delta^*$ , let  $x$  be the concatenation of the input components of the transitions in  $y$ .  $M'$  on input  $y$ , writes  $x$  on the stack while simulating the computation of  $M$  on  $x$  using the counters, and accepts in state  $f$  if  $M$  accepts  $x$ .  $M'$  is indeed deterministic since each symbol of  $\Delta$  implies the transition to apply.

Clearly,  $S(M')$  contains all strings of the form  $fw$ , where  $w$  is in  $L(M)$ . Let  $R$  be the regular language  $f\Sigma^*$ . Then  $L = S(M') \cap R$ .  $\square$

Therefore, this removal of nondeterminism with Proposition 12 is not always possible.

Lastly, two results will be stated that are shown below in Section 4.1 and Section 4.2 respectively, which are results on one-way stack automata, but require results on two-way automata for their proofs. The first is already known [7].

**Proposition 14.** [7] *If  $M$  is a one-way stack automaton,  $S(M) \in \mathcal{L}(\text{REG})$ .*

**Proposition 15.** *If  $M \in \text{NSCM}$  with only one 1-reversal-bounded counter over a unary alphabet, then  $S(M) \notin \mathcal{L}(\text{NPCM})$ .*

The latter result is interesting in the following sense: An NPCM combines a pushdown and reversal-bounded counters; a pushdown alone yields only regular store languages; NCM yields NCM store languages; their combination (NPCM) also yields NCM languages. An NSCM combines a stack and reversal-bounded counters. A stack alone yields only regular store languages; but, unlike, NPCM, the store languages of NSCM are more general, describing some languages that are not in NCM nor NPCM. However, it is seen next that NSCM machines only yield NSCM store languages.

**Proposition 16.** *If  $M \in \text{NSCM}$ , then  $S(M) \in \mathcal{L}(\text{NSCM})$ .*

PROOF. Let  $M$  be a  $k$ -counter NSCM machine. Construct a  $2k + 2$  counter NSCM machine (give names  $c_i, d_i, e, f$  to the counters,  $1 \leq i \leq k$ ) that simulates  $M$  with two identical copies of each counter,  $c_i$  and  $d_i$ ,  $1 \leq i \leq k$ , on a guessed input. Then, at some nondeterministically guessed spot,  $M'$  verifies that the stack contents are the same as the input by moving the stack head to the left-end-marker while adding one to counter  $e$  and  $f$ , then  $M'$  verifies that the stack contents are the same as the input by comparing the stack to the input symbol-by-symbol while decreasing  $e$  to verify that the read/write head on the input is in the correct location. Then  $M'$  returns its stack read head to the proper location using counter  $f$ . Then  $M'$  verifies that the counter values match the input values by decreasing each  $c_i$ . Then  $M'$  continues the simulation on the second set of counters,  $d_i$ , accepting if  $M$  accepts.  $\square$

### 3.2. Connections between Deterministic and Nondeterministic Machines

Thus far, the primary concern has been store languages of nondeterministic machine models. In this section, a connection between deterministic and nondeterministic one-way machines is demonstrated.

**Proposition 17.** *Let  $\Omega_1, \dots, \Omega_k$  be store types, and let  $M$  be a one-way nondeterministic  $(\Omega_1, \dots, \Omega_k)$ -machine. One can construct a one-way deterministic  $(\Omega_1, \dots, \Omega_k)$ -machine acceptor  $M'$  of the same type as  $M$  such that  $S(M') = S(M)$ .*

PROOF. Let  $t_1, \dots, t_m$  be new symbols in bijective correspondence with the transitions of  $M$ . Then let  $M'$  operate as follows over the input alphabet  $t_1, \dots, t_m$ .  $M'$  reads input symbol  $t_i$  and simulates  $t_i$  on the store (while ignoring the input symbol of  $M$  in  $t_i$  and switching states identically according to the transition symbols). Then,  $M'$  accepts on the right end-marker if the simulated state is final. Then the store of  $M'$  is changed identically to  $M$ , and so  $S(M') = S(M)$ .  $\square$

**Corollary 18.**  $S(\text{NPDA}) = S(\text{DPDA}), S(\text{NCM}) = S(\text{DCM})$

This is also true for all one-way nondeterministic and deterministic machine models considered in this paper. Note that the following have been observed:

1. If  $M$  is an NFA or an NPDA, then  $S(M)$  is regular and hence the deterministic version of the model can accept its own store language.
2. If  $M$  is an NCM, then  $S(M)$  can be accepted by a DCM, hence the deterministic version of the model can accept its own store language.

However, it follows from previous analyzes that the above does not hold for DPCMs:

**Proposition 19.** *There is a 0-reversal-bounded DPCM  $M$  such that  $S(M)$  cannot be accepted by any DPCM.*

PROOF. Suppose otherwise. It is known that there are languages in NCM that are not in DPCM [25]. Let  $L$  be such a language. By Proposition 13, there exists  $M' \in \text{DPCM}$  that is 0-reversal-bounded such that  $(fZ_0)^{-1}S(M') = L(M)$ . But  $S(M') \in \text{DPCM}$  by the assumption, and since it is clear that DPCM is closed under left quotient with a fixed word,  $L(M)$  is as well, a contradiction.  $\square$

#### 4. Machines with Two-Way Read-Only Inputs

Using exactly the same store types as defined in the previous section, two-way input machines can also be defined. Given store types  $\Omega_1, \dots, \Omega_k$  with  $\Omega_i = (\Gamma_i, I_i, f_i, g_i, c_{0,i}, L_{I,i}), 1 \leq i \leq k$ , two-way inputs have an end-marker on both sides,  $\triangleright w \triangleleft$ , and the transition function is a mapping from  $Q \times (\Sigma \cup \{\triangleright, \triangleleft\}) \times \Gamma_1 \times \dots \times \Gamma_k$  to  $Q \times I \times I_1 \times \dots \times I_k \times \{-1, 0, +1\}$  (describing the direction of the input head movement), a configuration of  $M$  is a quadruple  $(q, \triangleright w \triangleleft, \gamma, j)$ , where  $q \in Q, w \in \Sigma^*, \gamma \in \Gamma_0^*, 1 \leq i \leq |w| + 3$  giving the current position on the input (the last position is off the input tape). The derivation relation  $\vdash_M$  is defined by:  $(q, \triangleright w \triangleleft, \gamma_1, \dots, \gamma_k, j) \vdash_M (q', \triangleright w \triangleleft, \gamma'_1, \dots, \gamma'_k, j')$  if there exists  $(q', \iota_1, \dots, \iota_k, n) \in \delta(q, a, d_1, \dots, d_k)$ ,  $a$  is the  $j$ th character of  $\triangleright w \triangleleft$ ,  $j' = j + n, g(\gamma_i) = d_i f(\gamma_i, \iota_i) = \gamma'_i$ . Validity is defined just like with one-way machines. The language accepted by  $M$ ,  $L(M) = \{w \mid (q_0, \triangleright w \triangleleft, c_{0,1}, \dots, c_{0,k}, 1) \vdash_M^x (q_f, \triangleright w \triangleleft, \gamma_1, \dots, \gamma_k, j), q_f \in F, w \in \Sigma^*, \gamma_i \in \Gamma_i^*, 1 \leq j \leq |w| + 3, x \text{ is valid}\}$ . The store language of  $M$ ,  $S(M)$  is equal to  $\{q\gamma_1 \dots \gamma_k \mid (q_0, \triangleright w \triangleleft, c_{0,1}, \dots, c_{0,k}, 1) \vdash_M^x (q, \triangleright w \triangleleft, \gamma_1, \dots, \gamma_k, j) \vdash_M^y (q_f, \triangleright w \triangleleft, \gamma'_1, \dots, \gamma'_k, j'), q \in Q, q_f \in F, \gamma_i, \gamma'_i \in \Gamma_i^*, j, j' \in \{1, \dots, |w| + 3\}, xy \text{ is valid}\}$ .

In the previous section, store languages of different types of machines with a one-way read-only input were studied. What happens if  $M$  has a two-way input (with end markers  $\triangleright$  and  $\triangleleft$  on both ends of the input tape)?

##### 4.1. Two-Way NCMs and DCMs

This subsection considers store languages of two-way NCM (2NCM) and two-way DCM (2DCM). A machine is *finite-crossing* if there is a  $d \in \mathbb{N}$  such that in any computation, the input head crosses the boundary between any two adjacent cells of the input no more than  $d$  times. The first result demonstrates the surprising fact that the store languages of finite-crossing 2NCM can always be accepted by machines that are only one-way deterministic.

**Proposition 20.** *If  $M$  is a finite-crossing 2NCM, then  $S(M) \in \mathcal{L}(\text{DCM})$ .*

PROOF. Given  $k$ -counter  $M$  over  $\Sigma$ , first, construct a finite-crossing 2NCM  $M_1$  with  $2k + 1$  counters and input of the form  $xc_1^{i_1} \dots c_k^{i_k}$ , where  $x \in \Sigma^*$ .

$M_1$  simulates the computation of  $M$  on  $x$  with two sets of counters (named  $C_i$  and  $D_i$  for  $1 \leq i \leq k$ ). At some point nondeterministically chosen,  $M_1$  stores the input head position in the remaining counter, checks that the input segment  $qc_1^{i_1} \# \dots \# c_k^{i_k}$  corresponds to the state and the values of the  $D$  counters. If so,  $M_1$  continues the simulation of  $M$  on the correct position of  $x$  using the  $C$  counters, and accepts if and only if  $M$  accepts. It is known that all finite-crossing 2NCMs can be converted to one-way NCMs [23], and so convert  $M_1$  to an NCM  $M_2$  and then construct an NCM  $M_3$  that erases the  $x$  [1]. Then convert  $M_3$  to a DCM  $M_4$  as all bounded NCM languages are in DCM [24].  $\square$

Without the finite-crossing condition, this no longer holds.

**Proposition 21.** *There is a non-finite-crossing 2DCM  $M$  with only one 1-reversal counter over the bounded language  $a^*b^*$  such that  $S(M) \notin \mathcal{L}(\text{NPCM})$ .*

PROOF. Construct a 2DCM  $M$  with one 1-reversal counter which accepts  $\{a^i b^j \mid i, j > 1, i \text{ is a multiple of } j\}$  as follows:  $M$  stores  $i$  in counter  $C$  and enters a distinguished state  $f$ . (Thus, the configuration at this time is  $fa^i$ .) Then  $M$  changes state and checks (by decrementing  $C$  while going back-and-forth on  $b^j$ ) and accepts if  $i$  is divisible by  $j$ . Clearly,  $M$ 's counter makes only one reversal.

It follows from the construction that if  $i$  is divisible  $j$ , then  $fa^i$  would be a reachable configuration in some accepting computation.

Hence, among the words in  $S(M)$  are strings of the form  $fa^n$ , where  $n$  is composite.

If  $S(M)$  is in NPCM, then  $L' = S(M) \cap fa^* = \{fa^n \mid n \text{ is composite}\}$  can be accepted by an NPCM. This is a contradiction, since the Parikh map of  $L'$  is not semilinear, but it is known that the Parikh map of any NPCM language is semilinear [1].  $\square$

The next result was already mentioned in Section 3, but the proof appears here since it involves a proof using two-way machines.

**Proposition 22.** *There is a non-finite crossing 2NCM  $M$  with one 1-reversal counter over a one-letter alphabet such that  $S(M) \notin \mathcal{L}(\text{NPCM})$ .*

PROOF. Construct an  $M$  which first stores in counter  $C$ , a nondeterministically chosen number  $n$  and enter state  $f$ . Then it changes state and checks that  $n$  is a multiple of the unary input. Then, as in Proposition 21,  $S(M)$  is not in NPCM.  $\square$

**Proposition 23.** *If  $M \in \text{NSCM}$  with only one 1-reversal-bounded counter over a unary alphabet, then  $S(M) \notin \mathcal{L}(\text{NPCM})$ .*

PROOF. Let  $M$  be a 2NCM with one 1-reversal-bounded counter over a unary language such that  $S(M) \in \mathcal{L}(\text{NPCM})$ . Then, create a NSCM with one counter, where  $M'$  copies the input to the stack, and then simulates  $M$  on the stack contents and the counter. Assume  $S(M') \in \mathcal{L}(\text{NPCM})$ . Let  $g$  be a gsm that erases the stack contents, and ignores any state before the input is copied. Then  $g(S(M')) = S(M) \in \mathcal{L}(\text{NPCM})$  since this family is a full trio and is therefore closed under gsm mappings. But this contradicts Proposition 22.  $\square$

This section is concluded with a result that shows that for a particular two-way model of computation, the store languages can be more complex than the languages accepted.

**Proposition 24.**

1. *If  $M$  is a 2NCM over a unary input alphabet  $\{a\}$  (has left and right end markers), then  $L(M)$  is regular.*
2. *There is a 2NCM  $M$  augmented with one 1-reversal-bounded counter over a unary input alphabet  $\{a\}$  such that the Parikh map of  $S(M)$  is not semilinear (hence, not regular).*

PROOF. Part 1 was shown in [26]. For Part 2, construct  $M$  which, when given input  $a^i$  (with end markers) for some  $i \geq 1$ , first nondeterministically increments the counter by a value  $j \geq 1$  and enters a distinguished state  $f$ . Then it uses its two-way input to check that  $j$  is divisible by  $i$  and accepts. If  $S(M)$  is semilinear, then  $S(M) \cap fa^+ = \{fa^j \mid j \geq 1 \text{ is composite}\}$ , which is not semilinear.  $\square$

Here, the languages accepted by the machines are all regular, but the store languages are not even semilinear.

#### 4.2. Connections Between One-Way and Two-Way Machines

This section starts with the following straightforward lemma that shows that store languages of one-way nondeterministic machines are equivalent to those only accepting the empty word.

**Lemma 25.** *Let  $\Omega_1, \dots, \Omega_k$  be store types, and let  $\mathcal{M}$  be the set of one-way nondeterministic  $(\Omega_1, \dots, \Omega_k)$ -machines. If  $M \in \mathcal{M}$ , then there exists  $M' \in \mathcal{M}$  such that  $L(M') = \{\epsilon\}$  and  $S(M') = S(M)$ . Then the family  $\{S(M) \mid M \in \mathcal{M}\} = \{S(M) \mid M \in \mathcal{M}, L(M) = \{\epsilon\}\}$ .*

PROOF. Construct  $M'$  which, on  $\epsilon$  input, guesses and simulates the computation of  $M$  on some input  $x$  symbol-by-symbol. Since the sequence of ways the store can change is the same as in  $M$ , then  $M'$  must be a  $(\Omega_1, \dots, \Omega_k)$ -machine (i.e. in the definition of store types, all sequences of store instructions used in  $M$  in accepting computations are the same for  $M'$ , thereby being in  $L_I$ ).  $\square$

The above lemma is not true for deterministic machines  $M$ , since  $S(M)$  may be infinite, but  $S(M')$  for any deterministic machine  $M'$  is always finite.

Next, a connection will be demonstrated between sets of one-way and two-way machines of the same store type.

For example, if  $\mathcal{M}_1$  is the class of NPDAs with  $k$  reversal-bounded counters, then  $\mathcal{M}_2$  is the class of 2NPDAs with  $k$  reversal-bounded counters.

**Proposition 26.** *Let  $\Omega_1, \dots, \Omega_k$  be store types, let  $\mathcal{M}_1$  be the set of one-way nondeterministic  $(\Omega_1, \dots, \Omega_k)$ -machines and let  $\mathcal{M}_2$  be the set of two-way nondeterministic  $(\Omega_1, \dots, \Omega_k)$ -machines. Then  $\{S(M) \mid M \in \mathcal{M}_1\} = \{S(M) \mid M \in \mathcal{M}_2, L(M) \text{ finite}\}$ .*

PROOF. “ $\subseteq$ ” From Lemma 25,  $\{S(M) \mid M \in \mathcal{M}\} = \{S(M) \mid M \in \mathcal{M}, L(M) = \{\epsilon\}\}$ . Then, for every machine in the second set, a two-way machine can be constructed (on epsilon input and thus the two-way head never moves off end-markers).

“ $\supseteq$ ” Let  $M_2 \in \mathcal{M}_2$ . For each  $w \in L(M_2)$ , there exists  $S_w(M_2)$  consisting of all words  $x \in S(M_2)$  that can appear in an accepting computation on input  $w$ . Then  $\bigcup_{w \in L(M_2)} S_w(M_2) = S(M_2)$ .

Then  $\mathcal{L}(\mathcal{M}_2)$  must be closed under union, since, in the definition of store types, if two machines use valid sequences of instructions according to the instruction languages  $L_I$ , so must the obvious nondeterministic machine constructed to accept their union that goes from the initial state to subsequent configurations in only one of the machines (by keeping the states disjoint). Thus, it is sufficient to show that for each  $w \in L(M_2)$ ,  $S_w(M_2)$  is in  $\{S(M) \mid M \in \mathcal{M}_1\}$ .

Construct a machine  $M_1$  in  $\mathcal{M}_1$  such that  $S(M_1) = S_w(M_2)$ . Then  $M_1$  keeps  $w$  in its finite control.  $M_1$  on input  $\epsilon$  simulates the head movements of  $M_2$  on  $w$  (in the finite control, by keeping track of the simulated position of  $M_2$  on  $w$ ), while simulating the computation of  $M$  on the storage faithfully. Thus,  $M_1$  accepts  $\epsilon$  if and only if  $M_2$  accepts  $w$ . If the input to  $M_1$  is not  $\epsilon$ , it rejects. Clearly,  $S(M_1) = S_w(M_2)$ .  $\square$

As applications of the above, the following corollaries to the results already shown in Section 3 are obtained:

1. If  $M$  is a 2NPDA and  $L(M)$  is finite, then  $S(M)$  is regular.
2. If  $M$  is a 2NTM with reversal-bounded read/write tape and  $L(M)$  is finite, then  $S(M)$  is regular.
3. If  $M$  is a 2NTM with reversal-bounded read/write tape and reversal-bounded counters and  $L(M)$  is finite, then  $S(M)$  is in  $\mathcal{L}(\text{NCM})$ .

Similar corollaries hold for the other machine models studied in Section 3.

Then, by Lemma 25 and Proposition 26:

**Corollary 27.** *Let  $\Omega_1, \dots, \Omega_k$  be store types, let  $\mathcal{M}_1$  be the set of all one-way nondeterministic  $(\Omega_1, \dots, \Omega_k)$ -machines and let  $\mathcal{M}_2$  be the set of all two-way nondeterministic  $(\Omega_1, \dots, \Omega_k)$ -machines.*

*The following language families are equal:*

1.  $\{S(M) \mid M \in \mathcal{M}_1\}$ ,
2.  $\{S(M) \mid M \in \mathcal{M}_1, L(M) = \{\epsilon\}\}$ ,
3.  $\{S(M) \mid M \in \mathcal{M}_2, L(M) = \{\epsilon\}\}$ ,
4.  $\{S(M) \mid M \in \mathcal{M}_2, L(M) \text{ finite}\}$ .

The assumption that  $L(M_2)$  is finite in the above Proposition 26 is necessary. For consider a 2DCA, which is a two-way deterministic machines with an unrestricted counter.

**Proposition 28.** *There is a 2DCA  $M$  which makes two sweeps on the input (left-to-right and then right-to-left, where acceptance is on the left end marker) and makes only  $O(\log n)$  reversals on the counter on input of size  $n$  such that its store language  $S(M)$  is non-regular. (If  $M$  makes only one sweep, then Proposition 4 shows that  $S(M)$  is regular.)*

PROOF. Construct  $M$  which, when given input  $w$ , operates as follows:

1.  $M$  makes a left-to-right sweep of the input  $w$  and checks that it is of the form

$$\triangleright a^{i_1} b^{j_1} a^{i_2} b^{j_2} \dots a^{i_k} b^{j_k} \triangleleft$$

for some  $k \geq 1, i_1, \dots, i_k, j_1, \dots, j_k \geq 1$ . It uses the counter to check that  $i_i = j_1, \dots, i_k = j_k$ . At the end of this process, the counter is zero.

2. Then  $M$  moves its input head left and increments the counter to value  $i_k (= j_k)$  and enter a *unique* state  $f$ . Thus the configuration of the counter and state at this time is  $f1^{i_k}$ . The state  $f$  is *only* entered at this time.
3. Next,  $M$  continues moving left checking that  $j_{k-1} = i_k/2, j_{k-2} = i_{k-1}/2, \dots, j_1 = i_2/2 = 1$  and accepts. (This is possible because there are two copies of  $i_k$  in each block.)

$S(M)$  is non-regular; otherwise  $S(M) \cap f1^+ = f\{1^{2^n} \mid n \geq 1\}$  would be regular. Clearly  $M$  is  $\log n$ -reversal-bounded on the counter.  $\square$

Next, the store language of two-way and one-way nondeterministic stack automata will be addressed. In [7], it was shown that the store language of a one-way stack automaton is regular. Here, an alternative simple proof of this result is provided by using the general connections established between one-way automata and two-way automata, and an existing result on two-way stack automata. In [5], it was shown that the set of all words that can appear in the store of a two-way stack automaton  $M$  on an input  $w \in \Sigma^*$  (not in general over all words, but over only a single word), when  $M$  “falls off” the right end-marker of  $w$ , is a regular language (this was used as a key step to showing all two-way stack languages are recursive). This fact will be combined with the results of this section to show that all store languages of one-way nondeterministic stack automata are regular. Two technical lemmas are required before a proof of the main result (essentially used to convert the notation used in [5] with our notation).

**Lemma 29.** *Let  $M$  be a two-way nondeterministic stack automaton. Then  $\{qx \downarrow y \mid (q_0, \triangleright, \triangleleft, \downarrow Z_0, 1) \vdash^* (q, \triangleright, \triangleleft, x \downarrow y, 1)\} \in \mathcal{L}(\text{REG})$ .*



PROOF. In [5], it is shown that, for each word  $\triangleright w \triangleleft$ , and each  $q \in Q$  then  $\{xqy \mid (q_0, \triangleright w \triangleleft, \downarrow Z_0, 1) \vdash^* (q, \triangleright w \triangleleft, x \downarrow y, |w| + 3)\}$  is a regular language. Then it is clear that, using the empty word (and any state),  $\{qx \downarrow y \mid (q_0, \triangleright \triangleleft, \downarrow Z_0, 1) \vdash^* (q, \triangleright \triangleleft, x \downarrow y, 3)\}$  is regular.

Let  $M'$  be a new machine that simulates  $M$ , but at every step, if the simulated  $M$  is in state  $q$ ,  $M'$  can nondeterministically move the input head past the right end-marker. Then  $\{qx \downarrow y \mid (q_0, \triangleright \triangleleft, \downarrow Z_0, 1) \vdash_{M'}^* (q, \triangleright \triangleleft, x \downarrow y, 3)\} = \{qx \downarrow y \mid (q_0, \triangleright \triangleleft, \downarrow Z_0, 1) \vdash_M^* (q, \triangleright \triangleleft, x \downarrow y, 1)\} \in \mathcal{L}(\text{REG})$ .  $\square$

**Lemma 30.** *Let  $M$  be a two-way nondeterministic stack automaton. Then  $\{qx \downarrow y \mid (q, \triangleright \triangleleft, x \downarrow y, 1) \vdash_M^* (q_f, \triangleright \triangleleft, z, 1), q_f \in F, z \in \Gamma^*\} \in \mathcal{L}(\text{REG})$ .*

PROOF. Let  $M'$  be a machine that does not ever move its input head, and on a new state  $q'_0$ , nondeterministically guesses a word  $z_1 \downarrow z_2$  and puts it on the stack, then  $M'$  nondeterministically switches to any final state of  $M$ . From there,  $M'$  simulates  $M$  in reverse. So, if  $M$  moves right in the storage, then  $M'$  moves left, if  $M$  moves left, then  $M'$  moves right. If  $M$  replaces  $x$  with  $b_1 \cdots b_m, m \geq 1$ , the  $M'$  pops  $b_m$  down to  $b_2$ , then replaces  $b_1$  with  $x$ , all on new states not in  $Q$ . If  $M$  pops  $x$ , then  $M'$  pushes  $x$ , etc.

Then,  $\{qx \downarrow y \mid (q_0, \triangleright \triangleleft, \downarrow Z_0, 1) \vdash_{M'}^* (q, \triangleright \triangleleft, x \downarrow y, 1)\}$  is regular by Lemma 29. This is equal to  $\{qx \downarrow y \mid (q'_0, \triangleright \triangleleft, \downarrow Z_0, 1) \vdash_{M'}^* (q'_0, \triangleright \triangleleft, z_1 \downarrow z_2, q) \vdash_{M'}^* (q_f, \triangleright \triangleleft, z, 1) \vdash_{M'}^* (q, \triangleright \triangleleft, x \downarrow y, 1)\}$ . Then this set intersected with  $Q\Gamma^*$  ( $Q$  does not contain any new states) is in  $\{qx \downarrow y \mid (q, \triangleright \triangleleft, x \downarrow y, 1) \vdash_M^* (q_f, \triangleright \triangleleft, z, 1), q_f \in F, z \in \Gamma^*\}$ , which must therefore be regular.  $\square$

By intersecting the two regular languages in the previous two lemmas, the following is obtained:

**Proposition 31.** *Let  $M$  be a two-way nondeterministic stack automaton such that  $L(M) = \{\epsilon\}$ . Then  $S(M)$  is regular.*

PROOF. From Lemmas 29 and 30, and since regular languages are closed under intersection,  $\{qx \downarrow y \mid (q_0, \triangleright \triangleleft, \downarrow Z_0, 1) \vdash^* (q, \triangleright \triangleleft, x \downarrow y, 1) \vdash^* (q_f, \triangleright \triangleleft, z, 1), q_f \in F, z \in \Gamma^*\} \in \mathcal{L}(\text{REG})$ .  $\square$

From this, and from Corollary 27, the following is obtained:

**Corollary 32.** *Let  $M$  be a two-way nondeterministic stack automaton such that  $L(M)$  is finite. Then  $S(M)$  is regular.*

**Corollary 33.** *Let  $M$  be a one-way nondeterministic stack automaton. Then  $S(M)$  is regular.*

Notice that this would be true for any type of two-way automata where the store language is regular over a fixed word.

## 5. Applications to Right Quotient

There are some nice applications of the results in this paper. One is addressed here.

A general proof is exhibited whereby it is shown that any deterministic automata class  $\mathcal{M}$  obtained from store types, where the nondeterministic machines with the same store types only accepts regular store languages, then  $\mathcal{M}$  is closed under right quotient with regular languages. This is perhaps surprising since right quotient seems to be quite difficult for deterministic machines.

**Definition 34.** *Let  $\Omega$  be a store type.  $\Omega$  is readable if the following are true:*

- *There are instructions to not change the store for any accessible store contents.*
- *At any point, it is possible to read the store one letter at a time, either from left-to-right (like a queue), or right-to-left (like a pushdown). It can be destructive (like a pushdown) or non-destructive (like a stack).*

The first condition is enforcing that it is possible to keep the same store contents. For example, with a pushdown automaton, it is always possible to replace the top of the pushdown  $X$  with  $X$ , thereby keeping it the same.

**Proposition 35.** *Let  $\Omega_1, \dots, \Omega_k$  be readable store types. Let  $\mathcal{M}_N$  be the set of all one-way nondeterministic  $(\Omega_1, \dots, \Omega_k)$ -machines, and let  $\mathcal{M}_D$  be the set of all one-way deterministic  $(\Omega_1, \dots, \Omega_k)$ -machines. If  $S(\mathcal{M}) \subseteq \text{REG}$ , then  $\mathcal{L}(\mathcal{M}_D)$  is closed under right quotient with regular languages.*

PROOF. Let  $M_1$  be a deterministic machine  $M_1 \in \mathcal{M}_D$  with state set  $Q$ . Let  $M_2$  be a DFA. A deterministic machine  $M_5 \in \mathcal{M}_D$  will be built accepting the right quotient of  $L(M_1)$  with  $L(M_2)$ .

First, build a new intermediate **nondeterministic** machine  $M_3 \in \mathcal{M}_N$  with states  $Q \cup Q' \cup Q''$  ( $Q, Q', Q''$  are disjoint; the primed states are explained below). It accepts the following language:

$$\{wx \mid wx \in L(M_1), x \in L(M_2)\}.$$

This is nondeterministic, because it must guess where the  $x$  starts. But,  $M_3$  can simulate a finite automaton in parallel nondeterministically. Also, build  $M_3$  such that when it hits the end of the  $w$  part of the input (but before the  $x$ ), if it's in state  $q$ , it switches to state  $q' \in Q'$ , then  $q'' \in Q''$  (requiring the store contents to not change between these configurations, which is possible by the first condition of the readable store type definition), then continues the simulation only using states from  $Q''$ .

Then, construct the store language  $S(M_3)$ . It is regular by the assumption. In fact, only words of  $S(M_3)$  that begin with  $Q'$  are needed. So consider  $S(M_3) \cap Q'\Gamma^*$ , and build a DFA  $M_4$  accepting this.

Now build a new **deterministic** machine  $M_5 \in \mathcal{M}_D$  that operates as follows. It simulates  $M_1$  on the input  $w$  until it hits the right input end-marker. At that point, say  $y$  is the contents of the store, and it is in state  $q$ . First, assume that machines in  $\mathcal{M}_N$  can read the store from left-to-right. Then read  $q'$  in the store language DFA  $M_4$  and see if  $q'y$  is in the store language deterministically on the store. If using a store that reads from right-to-left, instead use a DFA accepting  $S(M_4)^R$  instead of using  $S(M_4)$ . If  $M_4$  accepts  $q'y$ , then  $M_5$  accepts the input.

Let  $w \in L(M_3)$ . Then reading  $w$  in  $M_3$  (upon consuming the last letter) takes it to some configuration  $qy$ . Then  $qy \in L(M_3)$ , and so  $q'y$  is in the store language of  $M_3$ , which means that the machine  $M_3$  can accept from this configuration. And the fact that primed states are being used to enforce that it is at the right spot of the store language ensures that from that point on, the remaining word is in  $L(M_2)$ . Thus, there must be some  $x$  such that  $wx$  is in  $L(M_1)$  and  $x$  is in  $L(M_2)$ .

Conversely, if  $wx \in L(M_1)$  with  $x \in L(M_2)$ , then reading  $w$  in  $M_1$  takes it to some configuration  $qy$ . Then  $q'y$  must be in  $L(M_4)$ . Hence, by the construction of  $M_5$ ,  $w \in L(M_3)$ .

Hence,  $\mathcal{L}(\mathcal{M}_D)$  is closed under right quotient with regular languages.  $\square$

This implies regularity of several families, to our knowledge, three of which are known, and the rest unknown.

**Corollary 36.** *The following language families are closed under right quotient with regular languages:*

- *deterministic stack languages [14],*
- *deterministic checking stack languages,*
- *deterministic  $k$ -flip pushdown languages,*
- *deterministic pushdown automata [16],*
- *deterministic one counter automata [27],*
- *deterministic reversal-bounded queue automata,*
- *deterministic one-way read-only input deterministic Turing machines with a reversal-bounded worktape.*

This does provide an alternate, much shorter and more general proof, for stack and pushdown automata. It also resolves an explicitly stated unsolved open problem for  $k$ -flip pushdown automata [15]. All others are, to our knowledge, also unknown.

It is worth noticing the tight relationship between store languages and quotients. The intuition behind the closures under right quotient of all the families in Corollary 36 is that when the deterministic machines reach the end of their inputs, they can verify that their store contents are in the regular language constructed from the store language of a very similar nondeterministic machine. This same technique can even be true for non-regular store languages. For example, a similar technique could be used to show that DCM is closed under right quotient with NCM. This is because when the DCM reaches the end of its input, it only needs to verify that its store contents are in another NCM language, and the store language of an NCM language is in DCM. So it can do this in parallel with additional counters. However, in [27], a more general technique was used to show that DCM is closed under right quotient with even more general families such as NPDA and NPCM.

Note as well that not all deterministic families are closed under right quotient with regular languages, as DPCM is not [27]. Indeed, the store of a DPCM is not necessarily in DPCM, so when such a machine reaches the end of its input, there is not any way to verify that its store contents are “good” by using a store language within another DPCM machine.

## 6. Space Lower-Bounds for Non-Regular Store Languages of Turing Machines

In this section, the lower bounds will be studied on the space complexity of variations of the store language accepted by  $s(n)$  space-bounded NTMs and DTMs for the store language not to be regular. Here, 1NTM (1DTM) is used to denote a nondeterministic (deterministic) Turing machine with a one-way read-only input, and 2NTM (2DTM) is used to denote a nondeterministic (deterministic) Turing machine with a two-way read-only input.

A configuration of  $M$  is a tuple  $(q, \triangleright x \triangleleft, w, i)$ , where  $q$  is a state,  $x$  is the input with the input head on the  $i$ -th position, and the worktape has string  $w$  which includes the read/write head.

Let  $M$  be any such Turing machine with either a one-way or two-way read-only input and one read/write worktape (i.e., store) tape. The following two notions of  $M$  being  $s(n)$  space-bounded are used (see [28]):

1.  $M$  is strongly  $s(n)$  space-bounded if, for any input  $w$  of length  $n$ , all computations on  $w$  (accepting or not) use at most  $s(n)$  space on the worktape.
2.  $M$  is middle  $s(n)$  space-bounded if, for any input  $w$  of length  $n$  that is accepted, all accepting computations on  $w$  use at most  $s(n)$  space.

The following known results are needed:

### Proposition 37.

1.  $\log \log n$  is the lower bound for accepting non-regular languages by strongly (middle respectively) space-bounded 2NTMs and 2DTMs. [29, 30].
2.  $\log n$  is the lower bound for accepting non-regular languages by strongly (middle respectively) space-bounded 1NTMs and 1DTMs. [29].

In addition to the usual notion of the store language of space-bounded Turing machine, also the *strong store language* will be considered which is the set of reachable configurations. If  $M$  is a 2NTM (1NTM, 2DTM, 1DTM), the strong store language of  $M$  is  $S^s(M) = \{qw \mid \text{there is computation of } M \text{ (accepting or not) on some input of length } n \text{ that enters a configuration with state } q \text{ and } w \text{ on the worktape}\}$ .

**Proposition 38.** *If  $M$  is a middle  $s(n)$  space-bounded 2NTM and  $s(n)$  grows slower than  $\log \log n$ , then  $S(M)$  is regular.*

PROOF. Construct a 2NTM  $M'$  which, given an input  $wqx$ , where  $w$  is over the input alphabet of  $M$ ,  $q$  is a state, and  $x$  is over the worktape alphabet of  $M$  (assume that the state set and alphabets are distinct) operates as follows:

1.  $M'$  simulates  $M$  on  $w$ .
2. At some point nondeterministically chosen,  $M$  stops the simulation. Let the state and store contents of  $M$  (and, hence, also of  $M'$ ) at that time be  $q'$  and  $x'$ .  $M'$  converts  $x'$  to  $q'x'\#q'x'$ , where  $\#$  is a new symbol. (Thus  $M'$  makes two copies of  $q'x'$  separated by  $\#$ ).
3.  $M'$  then resumes the simulation of  $M$  using only the area to the right of  $\#$  in the worktape.
4. When  $M$  accepts,  $M'$  checks that  $qx$  on the input is identical to  $q'x'$  on the worktape and accepts.

Clearly  $M'$  is also strongly  $s(n)$  space-bounded, hence  $L(M')$  is regular by Proposition 37, part 1. Now, the strings in  $L(M')$  are of the form  $wqx$ . A homomorphism deleting  $w$  is then applied. It follows that the strong store language is regular.  $\square$

Furthermore, given a Turing machine  $M$  that is strongly  $s(n)$  space-bounded, one can build  $M'$  exactly like  $M$  but with all states final, and  $M'$  is middle  $s(n)$  space-bounded and  $S^s(M) = S(M')$ . Therefore:

**Corollary 39.** *If  $M$  is a strongly  $s(n)$  space-bounded 2NTM and  $s(n)$  grows slower than  $\log \log n$ , then  $S(M)$  is regular.*

Next, it will be shown that the  $\log \log n$  bound above is tight.

**Proposition 40.** *There is a strongly  $\log \log n$  space-bounded 2DTM  $M$  such that  $S^s(M)$  is not regular.*

PROOF. Let  $L = \{x_1\#x_2\#\dots\#x_k\# \mid k \geq 1, x_i \in \{0,1\}^*, x_i = 1, x_{i+1} = x_i + 1 \text{ for } 1 \leq i < k, x_k = 1^m \text{ for some } m\}$ . The addition is binary number addition. So, e.g.,  $1\#10\#11\#100\#101\#110\#111\#$  is in  $L$ . Construct a 2DTM  $M$  which, when given a string  $w$ , compares  $x_i$  and  $x_{i+1}$  bit-by-bit. When  $M$  determines that  $w$  is in  $L$ , the worktape will have  $1^m$  on its worktape.  $M$  then transforms this to  $1^m\#1^m$  and enters state  $f$ . Then  $S^s(M) \cap f1^+\#1^+ = \{f1^m\#1^m \mid m \geq 1\}$ , which is not regular. Hence  $S^s(M)$  is not regular.  $\square$

Hence, the following is immediate:

**Corollary 41.** *There is a middle  $\log \log n$  space-bounded 2DTM  $M$  such that  $S(M)$  is not regular.*

Turning now to one-way machines:

**Proposition 42.** *If  $M$  is a middle  $s(n)$  space-bounded 1NTM and  $s(n)$  grows slower than  $\log n$ , then  $S(M)$  is regular.*

PROOF. The proof is the same as the proof of Proposition 38 using Proposition 37, part 2, and noting that the  $M'$  constructed in that proof would also be one-way if  $M$  is one-way.  $\square$

**Corollary 43.** *If  $M$  is a strongly  $s(n)$  space-bounded 1NTM and  $s(n)$  grows slower than  $\log n$ , then  $S^s(M)$  is regular.*

The next result shows that Proposition 42 is tight.

**Proposition 44.** *There is a strongly  $\log n$  space-bounded 1DTM  $M$  such that  $S^s(M)$  is not regular.*

PROOF. Let  $L = \{a^n b^m \mid n \geq 1\}$ . Construct a strongly  $\log n$  space-bounded 1DTM to accept  $L$ .  $M$  when given an input  $a^n b^m$ , first reads  $a^n$  and stores  $n$  in binary, say  $x$ , on the worktape. Then  $M$  converts  $x$  to  $x\#x$ . Next,  $M$  reads  $b^m$  while decrementing the second  $x$  on the worktape to check that  $m = n$ . Finally,  $M'$  converts the worktape to  $x\#x$  and accepts in state  $f$ . Clearly,  $S^s(M) \cap f(0+1)^+\#(0+1)^+ = \{fz\#x \mid x \text{ is a binary encoding of a positive integer}\}$  is not regular. Hence,  $S^s(M)$  is not regular.  $\square$

**Corollary 45.** *There is a middle  $\log n$  space-bounded 1DTM  $M$  such that  $S(M)$  is not regular.*

## References

- [1] O. Ibarra, Reversal-bounded multicounter machines and their decision problems, *Journal of the ACM* 25 (1) (1978) 116–133.
- [2] J. Autebert, J. Berstel, L. Boasson, *Handbook of Formal Languages*, Vol. 1, Springer-Verlag, Berlin, 1997, Ch. Context-Free Languages and Pushdown Automata.
- [3] J. Eremondi, O. Ibarra, I. McQuillan, On the density of context-free and counter languages, in: I. Potapov (Ed.), *Lecture Notes in Computer Science*, Vol. 9168 of 19th International Conference on Developments in Language Theory, DLT 2015, 2015, pp. 228–239.
- [4] M. Holzer, M. Kutrib, Flip-pushdown automata: Nondeterminism is better than determinism, in: Z. Ésik, Z. Fülöp (Eds.), *Developments in Language Theory*, Vol. 2710 of *Lecture Notes in Computer Science*, 2003, pp. 361–372.
- [5] S. Ginsburg, S. Greibach, M. Harrison, Stack automata and compiling, *J. ACM* 14 (1) (1967) 172–201.
- [6] S. Ginsburg, S. Greibach, M. Harrison, One-way stack automata, *J. ACM* 14 (2) (1967) 389–418.
- [7] S. Bensch, J. Björklund, M. Kutrib, Deterministic stack transducers, in: Y.-S. Han, K. Salomaa (Eds.), *Implementation and Application of Automata: 21st International Conference, CIAA 2016, Seoul, South Korea, July 19–22, 2016, Proceedings*, Vol. 9705 of *Lecture Notes in Computer Science*, Springer International Publishing, 2016, pp. 27–38.
- [8] B. Baker, R. Book, Reversal-bounded multipushdown machines, *Journal of Computer and System Sciences* 8 (3) (1974) 315–332.
- [9] Z. Dang, Binary reachability analysis of pushdown timed automata with dense clocks, in: G. Berry, H. Comon, A. Finkel (Eds.), *Computer Aided Verification: 13th International Conference, CAV 2001, Proceedings*, 2001, pp. 506–517.
- [10] M. Hague, A. Lin, Model checking recursive programs with numeric data types, in: G. Gopalakrishnan, S. Qadeer (Eds.), *Computer Aided Verification*, Vol. 6806 of *Lecture Notes in Computer Science*, 2011, pp. 743–759.
- [11] O. Ibarra, J. Su, Z. Dang, T. Bultan, R. Kemmerer, Counter machines and verification problems, *Theoretical Computer Science* 289 (1) (2002) 165 – 189.
- [12] G. Paun, G. Rozenberg, A. Salomaa, *The Oxford Handbook of Membrane Computing*, Oxford University Press, Inc., New York, NY, USA, 2010.
- [13] G. Xie, Z. Dang, O. Ibarra, A solvable class of quadratic diophantine equations with applications to verification of infinite-state systems, in: J. Baeten, J. Lenstra, J. Parrow, G. Woeginger (Eds.), *Automata, Languages and Programming: 30th International Colloquium, ICALP 2003, Proceedings*, 2003, pp. 668–680.
- [14] J. Hopcroft, J. Ullman, Deterministic stack automata and the quotient operator, *Journal of Computer and System Sciences* 2 (1) (1968) 1 – 12.
- [15] P. Duris, M. Kosta, Flip-pushdown automata: nondeterministic  $\epsilon$ -moves can be removed, in: M. Lopatková (Ed.), *CEUR Workshop Proceedings: Proceedings of the Conference on Theory and Practice of Information Technologies*, Vol. 788 of *Information Technologies — Applications and Theory (ITAT)* 2011, 2011, pp. 15–22.
- [16] S. Ginsburg, S. Greibach, Deterministic context free languages, *Information and Control* 9 (6) (1966) 620–648.
- [17] J. E. Hopcroft, J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.
- [18] T. Harju, O. Ibarra, J. Karhumäki, A. Salomaa, Some decision problems concerning semilinearity and commutation, *Journal of Computer and System Sciences* 65 (2) (2002) 278–294.
- [19] S. Ginsburg, *Algebraic and Automata-Theoretic Properties of Formal Languages*, North-Holland Publishing Company, Amsterdam, 1975.
- [20] J. Engelfriet, H. Vogler, Look-ahead on pushdowns, *Information and Computation* 73 (3) (1987) 245 – 279.
- [21] O. Ibarra, I. McQuillan, The effect of end-markers on counter machines and commutativity, *Theoretical Computer Science* 627 (2016) 71–81.
- [22] J. Eremondi, O. Ibarra, I. McQuillan, Insertion operations on deterministic reversal-bounded counter machines, in: A. Dediu, E. Formenti, C. Martín-Vide, B. Truthe (Eds.), *Lecture Notes in Computer Science*, Vol. 8977 of 9th International Conference on Language and Automata Theory and Applications, LATA 2015, 2015, pp. 200–211.
- [23] E. Gurari, O. Ibarra, The complexity of decision problems for finite-turn multicounter machines, *Journal of Computer and System Sciences* 22 (2) (1981) 220–229.
- [24] O. Ibarra, S. Seki, Characterizations of bounded semilinear languages by one-way and two-way deterministic machines, *International Journal of Foundations of Computer Science* 23 (6) (2012) 1291–1306.
- [25] O. Ibarra, Visibly pushdown automata and transducers with counters, *Fundamenta Informaticae*. To appear. (2016).
- [26] O. Ibarra, T. Jiang, N. Tran, H. Wang, New decidability results concerning two-way counter machines, *SIAM J. Comput.* 23 (1) (1995) 123–137.
- [27] J. Eremondi, O. Ibarra, I. McQuillan, Deletion operations on deterministic families of automata, *Information and Computation*. To appear. (2016).
- [28] A. Szepietowski, *Turing Machines with Sublogarithmic Space*, Vol. 843 of *Lecture Notes in Computer Science*, Springer-Verlag Berlin Heidelberg, Berlin, 1994.
- [29] R. E. Stearns, J. Hartmanis, P. M. Lewis, Hierarchies of memory limited computations, in: *Proceedings of the 6th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965)*, FOCS '65, IEEE Computer Society, Washington, DC, USA, 1965, pp. 179–190. doi:10.1109/FOCS.1965.11. URL <http://dx.doi.org/10.1109/FOCS.1965.11>
- [30] J. E. Hopcroft, J. D. Ullman, Some results on tape-bounded turing machines, *J. ACM* 16 (1) (1969) 168–177.